1   1. (Currently Amended) An apparatus comprising:

2   at least one processor;

3   a memory coupled to the at least one processor;

4   a plurality of object oriented classes residing in the memory, at least one of the

5   plurality of object oriented classes including state data that indicates a protected class;

6   a catalog of allowed classes residing in the memory, the catalog of allowed classes

7   including all classes that are authorized to access at least one protected class; and

8   a state/domain checker residing in the memory and executed by the at least one

9   processor, the state/domain checker performing a plurality of checks when each of the

10   plurality of object oriented classes is loaded, the plurality of checks determining whether

11   a class being loaded accesses the at least one protected class, and if so, determining

12   whether the class being loaded is [authorized to access the at least one protected class]

13   included in the catalog of allowed classes, and generating an exception if the class being

14   loaded is not [authorized to access the at least one protected class] included in the catalog

15   of allowed classes.

1   2. (Original) The apparatus of claim 1 wherein the state/domain checker further performs

2   at least one runtime check when a method that may reference a dynamically defined class

3   is invoked and when a function is invoked that could potentially access a method on one

4   or more of the plurality of classes.

1   3. (Currently Amended) [The apparatus of claim 2] An apparatus comprising:

2   at least one processor;

3   a memory coupled to the at least one processor;

4   a plurality of object oriented classes residing in the memory, at least one of the

5   plurality of object oriented classes including state data that indicates a protected class;

6   a state/domain checker residing in the memory and executed by the at least one

7   processor, the state/domain checker performing a plurality of checks when each of the

8   plurality of object oriented classes is loaded, the plurality of checks determining whether

9   a class being loaded accesses at least one protected class, and if so, determining whether

10   the class being loaded is authorized to access the at least one protected class, and

11   generating an exception if the class being loaded is not authorized to access the at least

12   one protected class, wherein the state/domain checker further performs at least one

13   runtime check when a method that may reference a dynamically defined class is invoked

14   and when a function is invoked that could potentially access a method on one or more of

15   the plurality of classes, wherein the at least one runtime check includes a check to

16   determine whether a Java reflection method is invoked by a referencing class on a

17   referenced class, and if a Java reflection method is invoked by the referencing class, and

18   if the referenced class implements a private domain interface, and if the referencing class

19   does not implement a system state interface, generating an error.

1   4. (Currently Amended) [The apparatus of claim 2] An apparatus comprising:

2   at least one processor;

3   a memory coupled to the at least one processor;

4   a plurality of object oriented classes residing in the memory, at least one of the

5   plurality of object oriented classes including state data that indicates a protected class;

6   a state/domain checker residing in the memory and executed by the at least one

7   processor, the state/domain checker performing a plurality of checks when each of the

8   plurality of object oriented classes is loaded, the plurality of checks determining whether

9   a class being loaded accesses at least one protected class, and if so, determining whether

10   the class being loaded is authorized to access the at least one protected class, and

11   generating an exception if the class being loaded is not authorized to access the at least

12   one protected class, wherein the state/domain checker further performs at least one

13   runtime check when a method that may reference a dynamically defined class is invoked

14   and when a function is invoked that could potentially access a method on one or more of

15   the plurality of classes, wherein the at least one runtime check includes a check to

16   determine whether a Java Native Interface (JNI) function is invoked by an external

17   program to access a protected class, and if the external program invokes a JNI function to

18   access a protected class, and the external program is not running in system state,

19   generating an error.

1   5. (Original) The apparatus of claim 1 wherein a class is a protected class if the class is

2   defined as a private domain class or a system state class.

1   6. (Currently Amended) The apparatus of claim 1 wherein the plurality of checks includes

2   a check during class verification that determines whether a class being verified

3   implements a private domain interface or a system state interface, and if the class being

4   verified implements a private domain interface or a system state interface, and if the class

5   being verified is not included in [a] the catalog of allowed classes, generating an error.

1   7. (Currently Amended) The apparatus of claim [6] 1 wherein the catalog of [allowable]

2   allowed classes is generated during a build process that packages the plurality of classes

3   together into an installable form.

1   8. (Original) The apparatus of claim 1 wherein the plurality of checks includes a check

2   during class preparation that determines whether a class being prepared has a superclass,

3   and if the class being prepared has a superclass, and if the superclass implements a

4   private domain interface or a system state interface, and if the class being prepared does

5   not implement at least the same private domain interface or system state interface as the

6   superclass, generating an error.

1   9. (Original) The apparatus of claim 1 wherein the plurality of checks includes a check

2   during class resolution that determines whether a class being resolved to by a referencing

3   class implements a private domain interface, and if the class being resolved to by the

4   referencing class implements the private domain interface, and if the referencing class

5   does not implement a system state interface, generating an error.

1   10. (Original) The apparatus of claim 9 wherein the check during class resolution is

2   performed before runtime when a class is loaded.

1   11. (Original) The apparatus of claim 9 wherein the check during class resolution is

2   performed at runtime when a method on the class being resolved to is invoked.

1   12. (Original) An apparatus comprising:

2   at least one processor;

3   a memory coupled to the at least one processor;

4   a plurality of Java classes residing in the memory, at least one of the plurality of

5   Java classes including state data that indicates a protected class;

6   a Java Virtual Machine (JVM) residing in the memory and executed by the at least

7   one processor;

8   a state/domain checker residing in the memory and executed by the at least one

9   processor, the state/domain checker performing the following checks:

10   a first check during class verification that determines whether a class being

11   verified implements a private domain interface or a system state interface, and if

12   the class being verified implements a private domain interface or a system state

13   interface, and if the class being verified is not included in a catalog of allowed

14   classes that is generated during a JVM build process that packages the plurality of

15   classes together into an installable form, throwing an exception;

16   a second check during class preparation that determines whether a class

17   being prepared has a superclass, and if the class being prepared has a superclass,

18   and if the superclass implements a private domain interface or a system state

19   interface, and if the class being prepared does not implement at least the same

20   private domain interface or system state interface as the superclass, throwing an

21   exception;

22   a third check during class resolution that determines whether a class being

23   resolved to by a referencing class implements a private domain interface, and if

24   the class being resolved to by the referencing class implements the private domain

25   interface, and if the referencing class does not implement a system state interface,

26   throwing an exception;

(claim 12 continued)

27   a fourth check to determine whether a Java reflection method is invoked

28   by a referencing class on a referenced class, and if a Java reflection method is

29   invoked by the referencing class, and if the referenced class implements a private

30   domain interface, and if the referencing class does not implement a system state

31   interface, throwing an exception; and

32   a fifth check to determine whether a Java Native Interface (JNI) function is

33   invoked by a program external to the JVM to access a protected class at runtime,

34   and if the program invokes a JNI function to access a protected class, and the

35   program is not running in system state, throwing an exception.

1   13. (Currently Amended) A method for creating and enforcing protected system level

2   Java code comprising the steps of:

3   providing a catalog of allowed classes that includes all classes that are authorized

4   to access at least one protected class;

5   loading a plurality of Java classes, each of the plurality of Java classes that is

6   protected including state data that indicates a protected class; and

7   performing a plurality of checks when each of the plurality of Java classes is

8   loaded, the plurality of checks determining whether the class being loaded accesses at

9   least one protected class, and if so, determining whether the class being loaded is

10   [authorized to access the at least one protected class] included in the catalog of allowed

11   classes, and generating an exception if the class being loaded is not [authorized to access

12   the at least one protected class] included in the catalog of allowed classes.

1   14. (Original) The method of claim 13 further comprising the step of performing at least

2   one runtime check when a method that may reference a dynamically defined class is

3   invoked and when a function is invoked that could potentially access a method on one or

4   more of the plurality of classes.

1   15. (Currently Amended) [The method of claim 14] A method for creating and enforcing

2   protected system level Java code comprising the steps of:

3   loading a plurality of Java classes, each of the plurality of Java classes that is

4   protected including state data that indicates a protected class;

5   performing a plurality of checks when each of the plurality of Java classes is

6   loaded, the plurality of checks determining whether the class being loaded accesses at

7   least one protected class, and if so, determining whether the class being loaded is

8   authorized to access the at least one protected class, and generating an exception if the

9   class being loaded is not authorized to access the at least one protected class,

10   performing at least one runtime check when a method that may reference a

11   dynamically defined class is invoked and when a function is invoked that could

12   potentially access a method on one or more of the plurality of classes, wherein the at least

13   one runtime check includes a check to determine whether a Java reflection method is

14   invoked by a referencing class on a referenced class, and if a Java reflection method is

15   invoked by the referencing class, and if the referenced class implements a private domain

16   interface, and if the referencing class does not implement a system state interface,

17   generating an error.

1   16. (Currently Amended) [The method of claim 14] <u>A method for creating and enforcing</u>

2   <u>protected system level Java code comprising the steps of:</u>

3   <u>   loading a plurality of Java classes, each of the plurality of Java classes that is</u>

4   <u>protected including state data that indicates a protected class;</u>

5   <u>   performing a plurality of checks when each of the plurality of Java classes is</u>

6   <u>loaded, the plurality of checks determining whether the class being loaded accesses at</u>

7   <u>least one protected class, and if so, determining whether the class being loaded is</u>

8   <u>authorized to access the at least one protected class, and generating an exception if the</u>

9   <u>class being loaded is not authorized to access the at least one protected class,</u>

10   <u>   performing at least one runtime check when a method that may reference a</u>

11   <u>dynamically defined class is invoked and when a function is invoked that could</u>

12   <u>potentially access a method on one or more of the plurality of classes,</u> wherein the at least

13   one runtime check includes a check to determine whether a Java Native Interface (JNI)

14   function is invoked by an external program to access a protected class, and if the external

15   program invokes a JNI function to access a protected class, and the program is not

16   running in system state, generating an error.

1   17. (Original) The method of claim 13 wherein a class is a protected class if the class is

2   defined as a private domain class or a system state class.

1   18. (Currently Amended) The method of claim 13 wherein the plurality of checks

2   includes a check during class verification that determines whether a class being verified

3   implements a private domain interface or a system state interface, and if the class being

4   verified implements a private domain interface or a system state interface, and if the class

5   being verified is not included in [a] <u>the</u> catalog of allowed classes, generating an error.

1   19. (Currently Amended) The method of claim [18] <u>13</u> wherein the catalog of [allowable]

2   <u>allowed</u> classes is generated during a JVM build process that packages the plurality of

3   classes together into an installable form.

1   20. (Original) The method of claim 13 wherein the plurality of checks includes a check
2   during class preparation that determines whether a class being prepared has a superclass,
3   and if the class being prepared has a superclass, and if the superclass implements a
4   private domain interface or a system state interface, and if the class being prepared does
5   not implement at least the same private domain interface or system state interface as the
6   superclass, generating an error.

1   21. (Original) The method of claim 13 wherein the plurality of checks includes a check
2   during class resolution that determines whether a class being resolved to by a referencing
3   class implements a private domain interface, and if the class being resolved to by the
4   referencing class implements the private domain interface, and if the referencing class
5   does not implement a system state interface, generating an error.

1   22. (Original) The method of claim 21 wherein the check during class resolution is
2   performed before runtime when a class is loaded by a Java Virtual Machine (JVM).

1   23. (Original) The apparatus of claim 21 wherein the check during class resolution is
2   performed at runtime when a method on the class being resolved to is invoked.

1   24. (Original) A method for creating and enforcing protected system level Java code

2   comprising the steps of:

3   running a Java Virtual Machine (JVM);

4   the JVM loading a plurality of Java classes, each of the plurality of Java classes

5   that is protected including state data that indicates a protected class;

6   performing a first check during class verification that determines whether a class

7   being verified implements a private domain interface or a system state interface, and if

8   the class being verified implements a private domain interface or a system state interface,

9   and if the class being verified is not included in a catalog of allowed classes that is

10   generated during a JVM build process that packages the plurality of classes together into

11   an installable form, throwing an exception;

12   performing a second check during class preparation that determines whether a

13   class being prepared has a superclass, and if the class being prepared has a superclass, and

14   if the superclass implements a private domain interface or a system state interface, and if

15   the class being prepared does not implement at least the same private domain interface or

16   system state interface as the superclass, throwing an exception;

17   performing a third check during class resolution that determines whether a class

18   being resolved to by a referencing class implements a private domain interface, and if the

19   class being resolved to by the referencing class implements the private domain interface,

20   and if the referencing class does not implement a system state interface, throwing an

21   exception;

22   performing a fourth check to determine whether a Java reflection method is

23   invoked by a referencing class on a referenced class at runtime, and if a Java reflection

24   method is invoked by the referencing class, and if the referenced class implements a

25   private domain interface, and if the referencing class does not implement a system state

26   interface, throwing an exception; and

12

(claim 24 continued)

27   performing a fifth check to determine whether a Java Native Interface (JNI)

28   function is invoked by a program external to the JVM to access a protected class at

29   runtime, and if the program invokes a JNI function to access a protected class, and the

30   program is not running in system state, throwing an exception.

1   25. (Original) A method for building a computer program that includes system level code,

2   the method comprising the steps of:

3   generating Java source code for a plurality of object oriented classes, each of the

4   plurality of object oriented classes that is protected including state data defined in the

5   Java source code that indicates a protected class;

6   identifying from the Java source code for each object oriented class which of the

7   plurality of object oriented classes are protected;

8   compiling the Java source code for the plurality of object oriented classes, thereby

9   creating a plurality of class files that correspond to the plurality of object oriented classes;

10   creating a catalog of allowable classes, the catalog including all protected classes;

11   compiling source code for a Java Virtual Machine (JVM) to produce an

12   executable JVM;

13   creating installable media that includes the executable JVM, the catalog of

14   allowable classes, and the class files.

1   26. (Currently Amended) A program product comprising:

2   a state/domain checker that performs a plurality of checks when each of a plurality

3   of object oriented classes is loaded, the plurality of checks determining whether a class

4   being loaded accesses at least one protected class, and if so, determining whether the

5   class being loaded is [authorized to access the at least one protected class] included in a

6   catalog of allowed classes that includes all classes that are authorized to access at least

7   one protected class, and generating an exception if the class being loaded is not

8   [authorized to access the at least one protected class] included in the catalog of allowed

9   classes; and

10   signal bearing media bearing the state/domain checker.

1   27. (Original) The program product of claim 26 wherein said signal bearing media

2   comprises recordable media.

1   28. (Original) The program product of claim 26 wherein said signal bearing media

2   comprises transmission media.

1   29. (Original) The program product of claim 26 wherein the state/domain checker further

2   performs at least one runtime check when a method that may reference a dynamically

3   defined class is invoked and when a function is invoked that could potentially access a

4   method on one or more of the plurality of classes.

1   30. (Currently Amended) [The program product of claim 29] A program product

2   comprising:

3   a state/domain checker that performs a plurality of checks when each of a plurality

4   of object oriented classes is loaded, the plurality of checks determining whether a class

5   being loaded accesses at least one protected class, and if so, determining whether the

6   class being loaded is authorized to access the at least one protected class, and generating

7   an exception if the class being loaded is not authorized to access the at least one protected

8   class, wherein the state/domain checker further performs at least one runtime check when

9   a method that may reference a dynamically defined class is invoked and when a function

10 is invoked that could potentially access a method on one or more of the plurality of

11 classes, wherein the at least one runtime check includes a check to determine whether a

12 Java reflection method is invoked by a referencing class on a referenced class, and if a

13 Java reflection method is invoked by the referencing class, and if the referenced class

14 implements a private domain interface, and if the referencing class does not implement a

15 system state interface, generating an error.

1   31. (Currently Amended) [The program product of claim 29] <u>A program product</u>
2   <u>comprising:</u>
3   <u>a state/domain checker that performs a plurality of checks when each of a plurality</u>
4   <u>of object oriented classes is loaded, the plurality of checks determining whether a class</u>
5   <u>being loaded accesses at least one protected class, and if so, determining whether the</u>
6   <u>class being loaded is authorized to access the at least one protected class, and generating</u>
7   <u>an exception if the class being loaded is not authorized to access the at least one protected</u>
8   <u>class, wherein the state/domain checker further performs at least one runtime check when</u>
9   <u>a method that may reference a dynamically defined class is invoked and when a function</u>
10   <u>is invoked that could potentially access a method on one or more of the plurality of</u>
11   <u>classes,</u> wherein the at least one runtime check includes a check to determine whether a
12   Java Native Interface (JNI) function is invoked by an external program to access a
13   protected class, and if the external program invokes a JNI function to access a protected
14   class, and the external program is not running in system state, generating an error.

1   32. (Original) The program product of claim 26 wherein a class is a protected class if the
2   class is defined as a private domain class or a system state class.

1   33. (Currently Amended) The program product of claim 26 wherein the plurality of
2   checks includes a check during class verification that determines whether a class being
3   verified implements a private domain interface or a system state interface, and if the class
4   being verified implements a private domain interface or a system state interface, and if
5   the class being verified is not included in [a] <u>the</u> catalog of allowed classes, generating an
6   error.

1   34. (Currently Amended) The program product of claim [33] <u>26</u> wherein the catalog of
2   [allowable] <u>allowed</u> classes is generated during a build process that packages the plurality
3   of classes together into an installable form.

1   35. (Original) The program product of claim 26 wherein the plurality of checks includes a
2   check during class preparation that determines whether a class being prepared has a
3   superclass, and if the class being prepared has a superclass, and if the superclass
4   implements a private domain interface or a system state interface, and if the class being
5   prepared does not implement at least the same private domain interface or system state
6   interface as the superclass, generating an error.

1   36. (Original) The program product of claim 26 wherein the plurality of checks includes a
2   check during class resolution that determines whether a class being resolved to by a
3   referencing class implements a private domain interface, and if the class being resolved to
4   by the referencing class implements the private domain interface, and if the referencing
5   class does not implement a system state interface, generating an error.

1   37. (Original) The program product of claim 36 wherein the check during class resolution
2   is performed before runtime when a class is loaded.

1   38. (Original) The program product of claim 36 wherein the check during class resolution
2   is performed at runtime when a method on the class being resolved to is invoked.

1   39. (Original) A program product comprising:

2   (A) a plurality of Java classes, at least one of the plurality of Java classes

3   including state data that indicates a protected class;

4   (B) a Java Virtual Machine (JVM) executable program;

5   (C) a state/domain checker that performs the following checks:

6   (C1) a first check during class verification that determines whether a class

7   being verified implements a private domain interface or a system state interface,

8   and if the class being verified implements a private domain interface or a system

9   state interface, and if the class being verified is not included in a catalog of

10   allowed classes that is generated during a JVM build process that packages the

11   plurality of classes together into an installable form, throwing an exception;

12   (C2) a second check during class preparation that determines whether a

13   class being prepared has a superclass, and if the class being prepared has a

14   superclass, and if the superclass implements a private domain interface or a

15   system state interface, and if the class being prepared does not implement at least

16   the same private domain interface or system state interface as the superclass,

17   throwing an exception;

18   (C3) a third check during class resolution that determines whether a class

19   being resolved to by a referencing class implements a private domain interface,

20   and if the class being resolved to by the referencing class implements the private

21   domain interface, and if the referencing class does not implement a system state

22   interface, throwing an exception;

23   (C4) a fourth check to determine whether a Java reflection method is

24   invoked by a referencing class on a referenced class at runtime, and if a Java

25   reflection method is invoked by the referencing class, and if the referenced class

26   implements a private domain interface, and if the referencing class does not

27   implement a system state interface, throwing an exception;

18

(claim 39 continued)

28   (C5) a fifth check to determine whether a Java Native Interface (JNI)

29   function is invoked by a program external to the JVM to access a protected class

30   at runtime, and if the program invokes a JNI function to access a protected class,

31   and the program is not running in system state, throwing an exception; and

32   (D) signal bearing media bearing the plurality of Java classes, the JVM executable

33 program, and the state/domain checker.

1   40. (Original) The program product of claim 39 wherein said signal bearing media

2   comprises recordable media.

1   41. (Original) The program product of claim 39 wherein said signal bearing media

2   comprises transmission media.

STATUS OF THE CLAIMS

Claims 1-41 were originally filed in this patent application. In response to the first office action dated 9/11/03, an amendment was filed on 12/10/03 that amended the specification, but did not amend any of the claims as originally filed. In the pending office action, claims 1, 5, 6, 8, 13, 17, 18, 20, 25-28, 32, 33 and 35 were rejected under 35 U.S.C. §103(a) as being unpatentable over the admitted prior art (APA) in view of U.S. Patent No. 6,381,734 to Golde. Claims 2, 14 and 29 were rejected under 35 U.S.C. §103(a) as being unpatentable over APA in view of Gold and further in view of U.S. Patent No. 6,397,384 to Briggs. Claims 9-11, 21-23, and 36-38 were rejected under 35 U.S.C. §103(a) as being unpatentable over APA in view of Gold and further in view of U.S. Patent No. 5,966,701 to Fresko. Claims 12, 14 and 39-41 were allowed. Claims 3, 4, 7, 15, 16, 19, 30, 31 and 34 were objected to as being dependent upon a rejected base claim, but would be allowable if properly rewritten in independent form. In this amendment, claims 1, 3, 4, 6, 7, 13, 15, 16, 18, 19, 26, 30, 32, 33 and 34 have been amended. Claims 1-41 are currently pending.